For the reasons hereinafter set forth, applicants respectfully submit that the rejection of Claims 1-18, in view of the teachings of the cited references, should be withdrawn and this application be allowed.

Prior to discussing the reasons why applicants believe that the claims of this application are clearly allowable, a brief discussion of the present invention, followed by a brief discussion of the cited and applied references, are presented. The following discussions of applicants' invention and the cited and applied references are not provided to define the scope or interpretation of any of the claims of this application. Instead, these discussions are provided to help the United States Patent and Trademark Office better appreciate important claim distinctions discussed thereafter.

Summary of the Invention

The present invention is directed to methods, computer-readable medium having computer-executable instructions, and systems for patching computer application programs that are not compatible with the computer operating system executing the computer application program. More specifically, the methods, the computer-readable medium having computer-executable instructions, and the systems determine whether or not the computer application program is compatible with the computer operating system executing the computer application program. If the computer application program is determined to be incompatible with the computer operating system, a debugger is started to run and patch the computer application program.

In one embodiment of the invention, application compatibility with the operating system is determined by checking application identity information against a database containing a list of applications that are currently known to be incompatible with the operating system. If the computer application program is found in the database, a set of attributes is checked to determine if the particular version of the application is incompatible. If the checked attributes match the

ones in the database, the application is determined to be incompatible with the operating system. If the application is not found in the database, the application is determined to be compatible with the operating system. Based on the determination, the application is either run with or without a debugger. Applications that work correctly under the operating system are not required to go through an additional level of execution created by the debugger application. Thus, such applications execute faster.

In an exemplary embodiment of the invention, if an application is found to be incompatible with the operating system, the operating system starts a debugger application that, in turn, runs the incompatible application. Before loading the incompatible application, the debugger loads a dynamic link library (DLL) that contains patches for the incompatible functions of the application. Specifically, the DLL contains a list of breakpoints specifying the location where the application needs to be patched, together with the appropriate code required to patch the application.

In this exemplary embodiment of the invention, when the debugger loads the DLL, a function is called that sets the breakpoints within the incompatible application. A breakpoint may be specified by (1) the module name and function name; (2) the module name and a memory offset; or (3) an address. The debugger also implements a handler that is capable of modifying the application code that is incompatible. In one embodiment of the invention, the debugger handler is capable of modifying the application code that is incompatible. Preferably, the debugger handler is capable of reading and writing memory within the incompatible application. Therefore, the handler is capable of modifying the incompatible code or inserting code into the application to fix the problem code contained within the application. For example, when the debugger reaches a breakpoint within the application, the debugger may be called to merely skip a portion of the incompatible code, or the handler may rewrite a portion of the code to contain a certain value in order to make the application compatible with the operating system.

One of the benefits of the use of a debugger application to patch incompatible applications is that this arrangement is very robust. The debugger is capable of monitoring every step an application takes while it is executing. Because the amount of code required to patch an application is generally very small, such as 200 bytes, patches are readily accessible to a user either through a Web site or FTP site.

Summary of the Cited and Applied References

Summary of Stone

Stone teaches methods for in-memory patching of a program after the program has been loaded into its memory space. Because a criterion of in-memory patching is not to patch the operating system or the program itself, Stone attempts to find a way of gaining access to the program address space from another process. Stone teaches three in-memory patching approaches: the loader approach, the API-hook/debug approach, and a message hook approach which Stone has yet to provide details of. Of the three approaches, the API-hook/debugger approach (hereinafter "API-hook approach") is of interest as it is cited by the Office Action to reject claims in the present invention.

The API-hook approach is to hook an API call into a program. The API call can be implemented to perform as a user wishes. For example, the API call can be made to patch the target program or its DLL in memory. The essence of the API-hook approach is to stop the execution of a program (hereinafter "target program") right before it starts and to inject patch code into the target program's address space. In this approach, in-memory patching is achieved by swapping, prior to actually running the target program, a page in the process space of the target program with code that loads DLL files containing the patch functions and changes the address of the original API functions with those of the patch functions. As a result, whenever the target program decides to call a function, the corresponding patch function is executed.

Stone teaches using the debugger interface provided by a Microsoft Windows® operating system to stop a target program before the target program ever gets executed. See Stone, p. 7, ¶ 5. The debugger interface enables a process to be created in Debug mode, which hides various status breakpoints in the process. See Stone, p. 8, ¶ 1. The stopping of the target program occurs at the status breakpoint just before the target program is about to start execution. See Stone, p. 8, last paragraph.

Stone teaches subsequently reading out a page in the target program's address space, inserting code into the page. The inserted code enables the target program to load the DLL containing the patch functions, to find the process addresses of the patch functions, and to replace the addresses of the patched functions, i.e., the original functions, with those of the patch functions. See Stone, p. 9, ¶ 1. Stone teaches further inserting an instruction at the end of the page; the instruction, when executed, will cause a debug event. The debug event will enable a user to suspend the execution of the target program and to restore the original page. See Stone, p. 9, ¶ 2.

Stone teaches utilizing context switch mechanisms to switch between the new page and the original page. Stone teaches first reading and saving the context of the process thread of the target program, then enabling the thread to point to and execute the new page in the target program's process space. At the end of execution, the context of the thread is reset and the old page is restored. See Stone, p. 10, ¶ 1.

In summary, Stone teaches selecting a page in the target program's address space; saving the content of the page; and filling in the page with code that loads a DLL containing patch functions for the target program, that finds the process addresses of the patch functions, and that replaces the process addresses of the to-be-patch functions, i.e., the original functions, with those of the patch functions. All these actions occur prior to the execution of the target program. It is

only after the execution of code in the new page, i.e., after the target program has been patched, that a debug event occurs, upon which the old page is restored.

As a result, Stone does not teach determining whether a computer application program is compatible with a computer operating system executing the computer application program. Further, nowhere does Stone teach starting a debugger to run the incompatible application. Neither does Stone teach running the steps of the incompatible application through the debugger. Nor does Stone teach the debugger patching the incompatible application whenever a breakpoint has been reached. As described above, Stone teaches patching a program before a debug event occurs. Occurrence of the debug event does not initiate the patching. Occurrence of the debug event only enables the restoration of the page in memory that has been replaced by code conducting the patching.

Summary of Lillich

Lillich intends to address the shortcomings inherent in "program driven patching." In program driven patching predicaments, the patches are installed automatically and, in essence, blindly, by the operating system whenever a program requests that a patch be installed. While this satisfies the program's request to install the patch, it leaves the operating system and other programs running on the operating system exposed to the unknown faults of the new patch. Lillich considers program driven patching to have three specific faults. First, the operating system does not know who is doing the patching. Second, the operating system does not know what functions are patched. Third, the operating system cannot control the order in which patches are installed. Lillich also considers that a dynamic patch analyzer won't solve these three mentioned faults. Therefore, Lillich purportedly teaches an operating system architecture that provides a layer of patch integrity verification and a corresponding strategy for evaluating patches prior to installation.

Therefore, Lillich purportedly teaches mechanisms for determining whether a patch is applicable to a process to produce the desired result. Lillich does not teach determining whether or not a computer application program is compatible with a computer operating system executing the computer application program, as recited by the present invention. Nor does Lillich teach starting a debugger to run the incompatible application and patching the incompatible application whenever a preset breakpoint in the application has been reached.

Summary of Nowlin

Nowlin purportedly teaches a method that converts a computer program designed for one operating system to run on a second operating system without a recompilation. This invention is specifically geared toward converting a non-Windows® CE application program into a Windows® CE-compatible computer program.

The method allegedly taught by Nowlin includes (a) loading a computer program on a computer system having a non-Windows® CE operating system; (b) translating the computer program to run on another computer system having a Windows® CE operating system; and (c) after the translation is completed, transferring the converted computer program to the second computer system running a Windows® CE operating system.

The translation is accomplished by creating a translation layer on the first computer system. The translation layer communicates with a non-Windows® CE system using a non-Windows® CE system calling convention, and with a Windows® CE system using a Windows® CE system calling convention. The translation layer converts calls for a non-Windows® CE operating system to calls that are compatible with a Windows® CE operating system. The translation includes parsing executable and dynamic link libraries and using separate filters to translate each of these different file types. The translation includes modifying the header of a file so that a Windows® CE operating system recognizes a file as a valid Windows® CE file.

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC
1420 Fifth Avenue, Suite 2800
Seattle, Washington 98101
206.682.8100

In summary, Nowlin merely provides a mechanism of parsing a non-Windows® CE-compatible computer application and translating the necessary program code on the non-Windows® CE computer system so that the application can be run on a Windows® CE computer system without recompilation. Nowlin does not teach or even remotely suggest using a debugger, not to say using a debugger in the manner recited by the claimed invention. Nor does Nowlin specifically teach comparing an attribute of one computer application program against an attribute of other compatible computer application programs, as recited by the claimed invention.

The Claims Distinguished

The Office Action has failed to show, and the applicants are unable to find, where any of the cited and applied references, alone or in combination, disclose, teach or suggest the subject matter of the claimed invention. As the above summaries show, among other differences, none of the cited and applied references teaches, discloses, or suggests a debugger that runs an application that is incompatible with the operating system the application is on. Nor does any of the cited and applied references teach, disclose, or suggest a debugger that sets breakpoints, runs an application and patches the application whenever meeting a breakpoint.

A. Independent Claims 1, 7, and 13

The Office Action has failed to show, and applicants have been unable to find, where any of the cited and applied references teaches or even remotely suggests the subject matter of independent Claims 1, 7, and 13, the only independent claims in this application. Claim 1 is a method claim, Claim 7 is a computer-readable medium claim, and Claim 13 is a system claim. More specifically, Claim 1 is directed to a method for patching a computer application program including a plurality of executable steps; Claim 7 is directed to a computer-readable medium having computer-executable instructions for patching a computer application program including a plurality of executable steps; and Claim 13 is directed at a computer system for patching a

computer application program wherein the computer system is capable of running an application having a plurality of executable steps. Claims 1, 7, and 13 all recite:

(a) determining whether or not the computer application program is compatible with a computer operating system executing the computer application program; and

(b) if the computer application program is determined to be incompatible with the computer operating system ("incompatible application"), starting a debugger to run the incompatible application, the debugger:

(i) setting at least one breakpoint within the incompatible application indicating a stopping point for the debugger;

(ii) running the steps of the incompatible application through the debugger; and

(iii) patching the incompatible application whenever a breakpoint has been reached.

The subject matter is not taught or even remotely suggested by either Stone or Lillich, combined or alone.

The Office Action suggests that Stone teaches "running the steps of the incompatible application through the debugger; and patching the incompatible application whenever a breakpoint has been reached." See Office Action, p. 4. Applicants respectfully disagree. As the above summary of the Stone reference points out, Stone teaches stopping a target program before the target program is executed before replacing a page in the target program's process space with patch code. Stone teaches inserting an instruction at the end of the patch code; the execution of the instruction during the execution of the target program causes a debug event; the debug event suspends the execution of the target program; the original page is then restored to the process space of the target program. Therefore, Stone teaches patching the target program before the occurrence of a debug event. The occurrence of a debug event in Stone only induces the restoration of the original page in the target program's process space. The occurrence of the debug event does not induce the patching of the target program, as recited in Claims 1, 7, and 13.

The Office Action correctly concludes that Stone does not teach determining the compatibility of a computer application program with a computer operating system executing the computer application program. Applicants respectfully agree with the Office Action's conclusion. However, the Office Action further alleges that Stone's in-memory patching strongly implies incompatibility of the target program to the running operating system. See Office Action, pp. 4-5. Applicants strongly disagree with such an impermissible hindsight construction of the claimed invention. Nowhere does Stone suggest that the in-memory patching approaches are aimed for curing incompatibility between a target program and the operating system running the target program. As Stone states, "the whole idea here [the API-hook approach] is to hook an API-call, and make it perform to our desires." Stone also suggests to "let the API-call the program performs be surrounded by our code so that we can make it perform in every way we wish." See Stone, p. 5, ¶ 2. These statements in Stone suggest the patching is to customize an API call according to a user's desire or wish. Nowhere does Stone suggest that in-memory patching strongly implies an incompatibility between the target program and the operating system running the target program.

The Office Action alleges that Lillich makes up Stone's deficiency in failing to teach determining the compatibility of a computer application program with a computer operating system executing the computer application program. Applicants respectfully disagree. Lillich teaches an operating system architecture that provides a layer of patch integrity verification and a corresponding strategy for evaluating patches prior to installation. More specifically, Lillich teaches a method that analyzes the functionality of a patch to predict a result of a call made by a desired process to the given function after the call has been redirected to the patch. See Lillich, Col. 6, lines 7-48. Therefore, Lillich does not teach determining whether a computer application program is compatible with the operating system that executes the computer application

program. Furthermore, there is no teaching or suggestion in Stone and Lillich, taken alone or in combination, why it would be obvious to combine the individual teachings of these references.

As the Office Action correctly concludes, Stone does not teach patching the incompatible application whenever a breakpoint has been reached. As noted above, Stone teaches patching a program in memory before the program is being executed. Therefore nowhere does Stone teach patching the incompatible application while executing the application and whenever a breakpoint has been reached.

Thus, there is simply no teaching or suggestion in Stone or Lillich of the subject matter recited in Claims 1, 7, or 13. Hence, applicants submit that Claims 1, 7, and 13 are clearly allowable.

Since all of the other claims remaining in this application depend from Claims 1, 7, and 13, respectively, these claims are submitted to be allowable for at least the same reasons that Claims 1, 7, and 13 are allowable. Further, these claims are submitted to be allowable for additional reasons.

B. Dependent Claims 5, 11, and 17

Claims 5, 11, and 17 depend from Claims 1, 7, and 13, respectively. Each of Claims 5, 11, and 17 recites that setting at least one breakpoint within the incompatible application comprises (a) loading a debugger dynamic link library containing a list of breakpoints, each breakpoint having a handler having a set of instructions for patching the incompatible application; and (b) the debugger accessing the list of breakpoints from the debugger dynamic link library and setting the breakpoints within the incompatible application. The Office Action alleges that Claims 5, 11, and 17 are anticipated by the combined teaching of Stone and Lillich. Applicants respectfully disagree.

The Office Action alleges that Stone and Lillich combined teach "loading and executing a debugger containing a set or list of breakpoint . . . the debugger setting a set of breakpoints

within the application." See Office Action, page 6. Applicants respectfully disagree. First, Claims 5, 11, and 17 recite a debugger dynamic link library, not the debugger itself, containing a list of breakpoints. Second, neither Stone or Lillich teaches loading a debugger dynamic library containing a list of breakpoints, each breakpoint having a handler having a set of instructions for patching the incompatible application. Further, neither Stone nor Lillich teach the debugger accessing the list of breakpoints from the debugger dynamic link library and the settings of breakpoints within the incompatible application. The fact that Stone teaches using the debugger interface supplied by a Microsoft Windows® operating system and a set of status breakpoints hidden in most Windows® procedures are not equivalent to having a debugger dynamic link library containing a list of breakpoints having a handler having a set of instructions for patching the incompatible application. In summary, neither Stone nor Lillich teaches the subject matter recited in Claims 5, 11, and 17.

Furthermore, there is no teaching or suggestion in Stone and Lillich, taken alone or in combination, why it would be obvious to combine the individual teachings of these references. More importantly, as noted above, even if these references were combinable—which applicants categorically deny—the resulting combination would not anticipate the subject matter of Claims 5, 11, and 17, especially when the subject matter of these claims is considered in combination with the subject matter of Claims 1, 7, and 13, the claims from which these claims depend. As a result, applicants respectfully submit that Claims 5, 11, and 17 are clearly allowable for reasons in addition to the reasons why the claims from which these claims depend are allowable.

C. Dependent Claims 6, 12, and 18

Claims 6, 12, and 18 depend from Claims 1, 7, and 13, respectively. Each of Claims 6, 12, and 18 recites that patching the application whenever a breakpoint has been reached

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC
1420 Fifth Avenue, Suite 2800
Seattle, Washington 98101
206.682.8100

comprises (a) calling the handler associated with the breakpoint; and (b) patching the incompatible application based on the instructions within the handler.

The Office Action correctly concludes that Stone does not teach calling a handler associated with a breakpoint. The Office Action alleges that it is obvious to use a handler at a breakpoint event for performing a patch. Even if it is obvious to use a handler at a breakpoint event for performing a patch, as pointed out above when discussing Claims 1, 7, and 13, Stone does not teach patching an incompatible application whenever a breakpoint has been reached. Instead, Stone teaches, upon the occurrence of a debug event, replacing the page containing the patch code with the original page in the target program's process space. Therefore, Stone does not teach or even remotely suggest the subject matter of Claims 6, 12, and 18. More importantly, as noted above, Stone would not anticipate the subject matter of Claims 6, 12, and 18 when the subject matter of these claims is considered in combination with the subject matter of Claims 1, 7, and 13, the claims from which these claims depend. As a result, applicants respectfully submit that Claims 6, 12, and 18 are allowable for reasons in addition to the reasons why the claims from which these claims depend are allowable.

D. Dependent Claims 3, 9, and 15

Claims 3, 9, and 15 are dependent upon Claims 1, 7, and 13, respectively. Each of Claims 3, 9, and 15 recites that determining if the application is compatible or incompatible with the operating system comprises (a) determining if at least the one identifying attribute of a plurality of identifying attributes of a computer application program matches at least one identifying attribute of a plurality of identifying attributes of incompatible applications; and (b) if at least one of the identifying attributes matches, determining that the computer application program is incompatible, otherwise determining that the computer application program is compatible.

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC
1420 Fifth Avenue, Suite 2800
Seattle, Washington 98101
206.682.8100

The Office Action correctly concludes that Stone does not teach the subject matter recited in Claims 3, 9, and 15. However, the Office Action alleges that the concept of checking an identification attribute among other attributes of a suspicious application is disclosed by Lillich and Nowlin. The Office Action suggests that Lillich teaches this concept by using a binding manager to see if the symbols associated with a patch bind correctly into memory before investigating what patch is appropriate to fix the memory and enables the program to be linked for a given platform. Applicants respectfully disagree. Determining whether symbols associated with a patch bind correctly into memory is in nowhere the same as determining if an attribute of a computer application program matches that of an incompatible application. That is, determining the appropriateness of a patch is not the same as determining whether an application is incompatible with a computer operating system in which the application executes. A patch is considered only after an application has been determined to be incompatible.

The Office Action alleges that Nowlin's teaching of using a filter DLL to patch executables of non-Windows® CE application so to make them compatible for a Windows® CE platform also teaches the concept of checking an identification attribute among other attributes of an application. Applicants respectfully disagree. As suggested by the Office Action (Office Action, page 8, last paragraph), Nowlin's teaching may imply matching attributes of one OS against what would be expected from another OS. But nowhere does Nowlin teach or remotely suggest determining if at least the one identifying attribute of a plurality of identifying attributes of a computer application program matches at least one identifying attribute of a plurality of identifying attributes of incompatible applications.

Furthermore, even if Nowlin and Lillich teach the concept suggested by the Office Action, neither Stone, Lillich, nor Nowlin teach the specific claim limitations recited in Claims 3, 9, and 15. None of the cited references teach comparing an attribute of one computer application program against an attribute of other incompatible computer application programs.

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC
1420 Fifth Avenue, Suite 2800
Seattle, Washington 98101
206.682.8100

More importantly, as noted above, even if it is obvious for one of ordinary skill in the art to combine the teachings of the cited references—which applicants categorically deny—The combination still would not anticipate the subject matter of Claims 3, 9, and 15 when the subject matter of these claims is considered in combination with the subject matter of Claims 1, 7, and 13, the claims from which these claims depend. Consequently, applicants respectfully submit that Claims 3, 9, and 15 are allowable for reasons in addition to the reasons why Claims 1, 7, and 13 are allowable.

E. Dependent Claims 4, 10, and 16

Claims 4, 10, and 16 are dependent on Claims 3, 9, and 15, respectively. Each of Claims 4, 10, and 16 recites that determining if the application is compatible or incompatible with the operating system further comprises (a) storing identifying attributes of incompatible applications; and (b) retrieving at least one of the stored identifying attributes for determining if at least one identifying attribute of a plurality of identifying attributes of the computer application program matches at least one of the stored identifying attributes of the incompatible applications.

The Office Action acknowledges that Nowlin does not specifically teach storing and using attributes of incompatible applications for matching against the attributes of the target application. However, the Office Action alleges that Nowlin's matching stored attributes of a compatible target OS against an incompatible OS is disclosing the same idea. Applicants respectfully disagree. Nowhere does Nowlin specifically teach the specific subject matter disclosed in Claims 4, 10, and 16.

More importantly, as noted above, even if it is obvious for one of ordinary skill in the art to modify the teaching of Nowlin to the claimed limitations in Claims 4, 10, and 16—which applicants categorically deny—Nowlin's teaching still would not anticipate the subject matter of Claims 4, 10, and 16 when the subject matter of these claims is considered in combination with

the subject matter of Claims 1, 3, 7, 9, 13, and 15, the claims from which these claims depend. As a result, applicants respectfully submit that these claims are allowable for reasons in addition to the reasons why the claims from which these claims depend are allowable.

CONCLUSION

In view of the foregoing comments, applicants respectfully submit that all of the claims in this application are clearly allowable in view of the cited and applied references. Consequently, early and favorable action allowing these claims and passing this application to issue is respectfully solicited. If the Examiner has any questions, he is invited to contact applicants' attorney at the number set forth below.

Respectfully submitted,

CHRISTENSEN O'CONNOR
JOHNSON KINDNESS PLLC

Joy Y. Xiang
Registration No. 55,747
Direct Dial No. 206.695.1607

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the U.S. Postal Service in a sealed envelope as first class mail with postage thereon fully prepaid and addressed to Mail Stop Amendment, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on the below date.

Date: April 22, 2005

JYX:pg

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC
1420 Fifth Avenue, Suite 2800
Seattle, Washington 98101
206.682.8100